Reductions

Pierre-Alain Fouque
Complexity class NTIME

\[ \text{NTIME}(t(n)) = \{ L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine} \}. \]

\[ \text{NP} = \bigcup_k \text{NTIME}(n^k). \]

- The class NP is insensitive to the choice of reasonable non-deterministic computational model because all such models are polynomially equivalent.
Examples of problems in NP

• A clique in a undirected graph is a subgraph, wherein every two nodes are connected by an edge. A k-clique is a clique that contains k nodes. E.g. A graph with a 5-clique

• The clique problem is to determine whether a graph contains a clique of a specified size: CLIQUE = {<G,k> | G is an undirected graph with a k-clique}
CLIQUE is in NP

**Proof Idea**  The clique is the certificate.

**Proof**  The following is a verifier $V$ for CLIQUE.

$V =$ “On input $\langle \langle G, k \rangle, c \rangle$:
1. Test whether $c$ is a subgraph with $k$ nodes in $G$.
2. Test whether $G$ contains all edges connecting nodes in $c$.
3. If both pass, accept; otherwise, reject.”

**Alternative Proof**  If you prefer to think of NP in terms of nondeterministic polynomial time Turing machines, you may prove this theorem by giving one that decides CLIQUE. Observe the similarity between the two proofs.

$N =$ “On input $\langle G, k \rangle$, where $G$ is a graph:
1. Nondeterministically select a subset $c$ of $k$ nodes of $G$.
2. Test whether $G$ contains all edges connecting nodes in $c$.
3. If yes, accept; otherwise, reject.”
SUBSET-SUM Problem

**SUBSET-SUM** = \{ ⟨S, t⟩ | S = \{x_1, \ldots, x_k\}, and for some \\
{y_1, \ldots, y_l} ⊆ \{x_1, \ldots, x_k\}, we have \(\sum y_i = t\)\}.

For example, ⟨\{4, 11, 16, 21, 27\}, 25⟩ ∈ **SUBSET-SUM** because 4 + 21 = 25.
Note that \{x_1, \ldots, x_k\} and \{y_1, \ldots, y_l\} are considered to be **multisets** and so allow repetition of elements.

**THEOREM 7.25**  
**SUBSET-SUM** is in NP.

**PROOF IDEA**  The subset is the certificate.

**PROOF**  The following is a verifier \(V\) for **SUBSET-SUM**.

\(V = \) “On input \(⟨S, t⟩, c⟩:\)
1. Test whether \(c\) is a collection of numbers that sum to \(t\).
2. Test whether \(S\) contains all the numbers in \(c\).
3. If both pass, **accept**; otherwise, **reject**.”

The complement of CLIQUE and **SUBSET-SUM** are not obvious members of NP. Verifying that something is not present seems more difficult than verifying it is present.
The P versus NP question

• NP is the class of languages that are solvable in polynomial time on a Non-deterministic TM or whereby membership in the language can be checked in polynomial time.

• P is the class of languages where membership can be tested in polynomial time.

P = the class of languages for which membership can be decided quickly.
NP = the class of languages for which membership can be verified quickly.
P vs. NP?

\[
\text{NP} \supset \text{P} \supset \text{P} = \text{NP}
\]

**FIGURE 7.26**
One of these two possibilities is correct

The best deterministic method currently known for deciding languages in NP uses exponential time. In other words, we can prove that

\[
\text{NP} \subseteq \text{EXPTIME} = \bigcup_{k} \text{TIME}(2^{n^{k}}),
\]

but we don’t know whether NP is contained in a smaller deterministic time complexity class.
NP-completeness

• Important advance on the P vs. NP question came in the early 1970s with the work of Stephen Cook and Leonid Levin
• They discover that certain problems in NP whose individual complexity is related to that of the entire class
• If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable
• These problems are called NP-complete
• Theory: if we have a polynomial time algorithm for an NP-complete problem, P=NP
• Practice: Prevent wasting time searching a nonexistent polynomial time algorithm to solve a particular problem
The satisfiability problem

• Boolean variables can take values: TRUE (1) and FALSE (0)
• Boolean operations AND (\(\land\)), OR (\(\lor\)) and NOT (\(\neg\))
• Boolean formula: \(\phi=(\bar{a}\land y) \lor (a\land z)\)
• A boolean formula is satisfiable if some assignment of 0s and 1s to the variables makes the formula evaluate to 1

\[
SAT = \{ \langle \phi \rangle | \phi \text{ is a satisfiable Boolean formula} \}.
\]

Now we state a theorem that links the complexity of the SAT problem to the complexities of all problems in NP.

\[
SAT \in P \text{ iff } P=NP
\]
Polynomial time reducibility

**Definition 7.28**

A function $f : \Sigma^* \rightarrow \Sigma^*$ is a *polynomial time computable function* if some polynomial time Turing machine $M$ exists that halts with just $f(w)$ on its tape, when started on any input $w$.

**Definition 7.29**

Language $A$ is *polynomial time mapping reducible*, or simply *polynomial time reducible*, to language $B$, written $A \leq_P B$, if a polynomial time computable function $f : \Sigma^* \rightarrow \Sigma^*$ exists, where for every $w$,

$$w \in A \iff f(w) \in B.$$ 

The function $f$ is called the *polynomial time reduction* of $A$ to $B$. 
If $A \leq_P B$ and $B \in P$, then $A \in P$.

**Proof** Let $M$ be the polynomial time algorithm deciding $B$ and $f$ be the polynomial time reduction from $A$ to $B$. We describe a polynomial time algorithm $N$ deciding $A$ as follows.

$N = \text{"On input } w:\text{ }
    \begin{enumerate}
    \item Compute $f(w)$.
    \item Run $M$ on input $f(w)$ and output whatever $M$ outputs.
    \end{enumerate}$

We have $w \in A$ whenever $f(w) \in B$ because $f$ is a reduction from $A$ to $B$. Thus, $M$ accepts $f(w)$ whenever $w \in A$. Moreover, $N$ runs in polynomial time because each of its two stages runs in polynomial time. Note that stage 2 runs in polynomial time because the composition of two polynomials is a polynomial.
3-CNF

form. A literal is a Boolean variable or a negated Boolean variable, as in $x$ or $\overline{x}$. A clause is several literals connected with $\lor$s, as in $(x_1 \lor \overline{x}_2 \lor \overline{x}_3 \lor x_4)$. A Boolean formula is in conjunctive normal form, called a cnf-formula, if it comprises several clauses connected with $\land$s, as in

$$(x_1 \lor x_2 \lor x_3 \lor x_4) \land (x_3 \lor \overline{x}_5 \lor x_6) \land (x_3 \lor x_6).$$

It is a 3cnf-formula if all the clauses have three literals, as in

$$(x_1 \lor x_2 \lor x_3) \land (x_3 \lor \overline{x}_5 \lor x_6) \land (x_3 \lor \overline{x}_6 \lor x_4) \land (x_4 \lor x_5 \lor x_6).$$

Let $3SAT = \{ \langle \phi \rangle \mid \phi$ is a satisfiable 3cnf-formula$\}$. If an assignment satisfies a cnf-formula, each clause must contain at least one literal that evaluates to 1.

The following theorem presents a polynomial time reduction from the $3SAT$ problem to the CLIQUE problem.

**Theorem 7.32**

$3SAT$ is polynomial time reducible to $CLIQUE$. 
Proof

Let $f$ be a formula with $k$ clauses
We can generate a string $<G,k>$ where $G$ is an undirected graph and $k$ an integer
The nodes are labeled by the literals in the clauses.
There is an edge between each nodes in the clauses if there is no incompatibility.

**Theorem:** There is an assignment for $f$ if and only if there is a $k$-clique in $G$

$(\Rightarrow)$ If we have a valid assignment then we pick each valid variable in each clause and they form a valid clique.

$(\Leftarrow)$ If we have a valid $k$-clique, then we can put these variables to true and the formula is valid.

**Figure 7.33**
The graph that the reduction produces from

$$
\phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor x_2)
$$
Definition of NP-Completeness

**Definition 7.34**

A language $B$ is **NP-complete** if it satisfies two conditions:

1. $B$ is in NP, and
2. every $A$ in NP is polynomial time reducible to $B$.

**Theorem 7.35**

If $B$ is NP-complete and $B \in P$, then $P = NP$.

**Proof** This theorem follows directly from the definition of polynomial time reducibility.

**Theorem 7.36**

If $B$ is NP-complete and $B \leq_P C$ for $C$ in NP, then $C$ is NP-complete.

**Proof** We already know that $C$ is in NP, so we must show that every $A$ in NP is polynomial time reducible to $C$. Because $B$ is NP-complete, every language in NP is polynomial time reducible to $B$, and $B$ in turn is polynomial time reducible to $C$. Polynomial time reductions compose; that is, if $A$ is polynomial time reducible to $B$ and $B$ is polynomial time reducible to $C$, then $A$ is polynomial time reducible to $C$. Hence every language in NP is polynomial time reducible to $C$. 
SAT is NP-complete

SAT is in NP: a ND polynomial TM guesses the assignation and we can easily cheched it

Take any language A in NP and show that A is P-time reducible to SAT
Let N be a NDTM deciding A in $n^k$ (k constant)
A tableau is accepting if any row is accepting conf.
($n^k)^2$ cells in the tableau
Variables $x_{i,j,s}$ is 1 if cell[i,j]==s
Formula: $f_{\text{cell}}$ AND $f_{\text{start}}$ AND $f_{\text{move}}$ AND $f_{\text{accept}}$

$$
\phi_{\text{cell}} = \bigwedge_{1 \leq i,j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right) \land \left( \bigwedge_{s,t \in C \atop s \neq t} (x_{i,j,s} \lor \overline{x_{i,j,t}}) \right) \right].
$$

$$
\phi_{\text{start}} = x_{1,1,#} \land x_{1,2,q_0} \land x_{1,3,w_1} \land x_{1,4,w_2} \land \ldots \land x_{1,n+2,w_n} \land x_{1,n+3,\uparrow} \land \ldots \land x_{1,n^k-1,\uparrow} \land x_{1,n^k,#}.
$$

$$
\phi_{\text{accept}} = \bigvee_{1 \leq i,j \leq n^k} x_{i,j,q_{\text{accept}}}.\n$$
Legal Moves

It is possible to encode each legal moves based on the transition table using a small number of variables.

We can verify that the size of the formula is polynomial in $n$ ($2k$ is a constant in the exponent)

$$\phi_{\text{move}} = \bigwedge_{1 \leq i < n^k, 1 < j < n^k} \text{(the } (i, j)\text{-window is legal)}.$$
From SAT to 3SAT: 3SAT is NP-complete

• We can replace each clause with at most 3 variables
• If a clause contains more than 3 variables \((a_1 \text{ OR } a_2 \text{ OR } a_3 \text{ OR } a_4)\) is can be rewritten as \((a_1 \text{ OR } a_2 \text{ OR } z) \text{ AND } (\text{NOT}(z) \text{ OR } a_3 \text{ OR } a_4)\)
• More generally

\[
(a_1 \lor a_2 \lor \cdots \lor a_l),
\]

we can replace it with the \(l - 2\) clauses

\[
(a_1 \lor a_2 \lor z_1) \land (\overline{z_1} \lor a_3 \lor z_2) \land (\overline{z_2} \lor a_4 \lor z_3) \land \cdots \land (\overline{z_{l-3}} \lor a_{l-1} \lor a_l).
\]

• CLIQUE is NP-complete
• 2-SAT is not NP-complete
Other reductions: Vertex cover

\[ \text{VERTEX-COVER} = \{ (G, k) \mid G \text{ is an undirected graph that has a } k \text{-node vertex cover} \}. \]

**Theorem 7.44**

\( \text{VERTEX-COVER} \) is NP-complete.

1. Show that \( \text{VERTEX-COVER} \) is in NP
2. Show that \( \text{VERTEX-COVER} \) is complete
Reduction between 3SAT and Vertex cover

\[ \text{VERTEX-COVER} = \{ (G, k) \mid G \text{ is an undirected graph that has a } k\text{-node vertex cover} \} . \]

**Theorem 7.44**

\[ \text{VERTEX-COVER} \text{ is NP-complete.} \]

**Figure 7.45**

The graph that the reduction produces from
\[ \phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor x_2) \]
THEOREM 7.56

SUBSET-SUM is NP-complete.

\[
\begin{array}{cccc|ccc}
& 1 & 2 & 3 & \cdots & l & c_1 & c_2 & \cdots & c_k \\
y_1 & 1 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\
z_1 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\
y_2 & 1 & 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\
z_2 & 1 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\
y_3 & 1 & 0 & \cdots & 0 & 1 & 1 & \cdots & 0 \\
z_3 & 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
y_l & 1 & 0 & 0 & \cdots & 0 \\
z_l & 1 & 0 & 0 & \cdots & 0 \\
g_1 & & & & & 1 & 0 & \cdots & 0 \\
h_1 & & & & & 1 & 0 & \cdots & 0 \\
g_2 & & & & & 1 & \cdots & 0 \\
h_2 & & & & & 1 & \cdots & 0 \\
\vdots & & & & & \ddots & & & \ddots & & \ddots & \\
g_k & & & & & & & & 1 \\
h_k & & & & & & & & 1 \\
t & 1 & 1 & 1 & 1 & \cdots & 1 & 3 & 3 & \cdots & 3 \\
\end{array}
\]

SURE 7.57

ducing 3SAT to SUBSET-SUM
**THEOREM 7.56**

*SUBSET-SUM* is NP-complete.

1. Show that SUBSET-SUM is in NP
2. Show that SUBSET-SUM is complete
Lectures


• Richard Karp gave the first 21 NP-complete problems

• It is useful to read such results to know which problems are very hard and it is useless to find a polynomial-time algorithm